OCR
Oxford Cambridge and RSA

An OCR endorsed teaching and learning tool

OCR A Level Computer Science

H446 – Paper 1

4

Programming language translators

Unit 2

Systems software and applications generation

PG ONLINE

Objectives

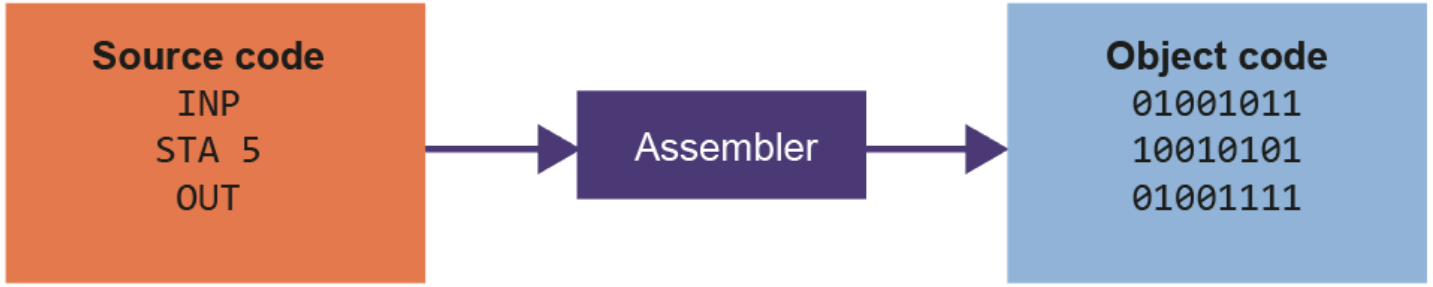
- Understand the role of an assembler, compiler and interpreter
- Explain the difference between compilation and interpretation, and describe situations when each would be appropriate
- Explain why an intermediate language such as bytecode is produced as the final output by some compilers and how it is subsequently used
- Describe the stages of compilation: lexical analysis, syntax analysis, code generation and optimisation
- Describe the function of linkers and loaders
- Describe the use of libraries

Assembly code

- Computers execute machine code
- It is difficult for humans to read, write and debug machine code
  - A machine code instruction might look like this: 01000101
- Assembly code instructions are equivalent to machine code but easier for humans to work with
  - An assembly code instruction might look like this: LDA 5

Assembler

- Assembly code is a low level language
- Translating assembly code instructions into machine code is done by an assembler
- Each processor has its own instruction set and so the object code produced will be hardware specific

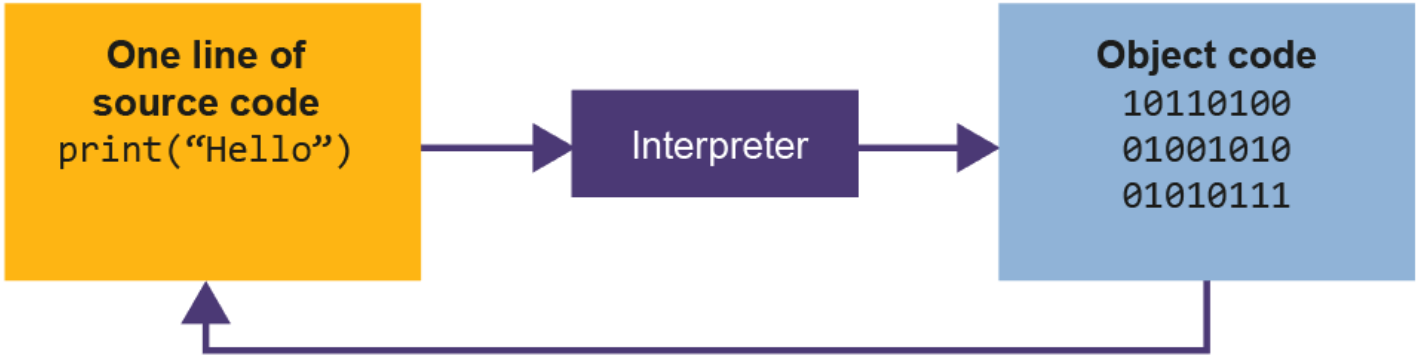


Compiler

- A compiler translates a whole program written in a high level language into executable machine code, going through several stages
  - Compiled high level languages include Visual Basic and C++
- The resulting machine code is called object code
- T
  s

Interpreter

- An interpreter also translates code written in a high level language into machine code
  - Interpreted high level languages include JavaScript and PHP
- However, the interpreter does this line by line rather than translating the whole program before any of it can be executed



Compiler vs Interpreter

- What do you think the advantages might be?

Compiler	Interpreter

Compiler vs Interpreter

- What do you think the advantages might be?

Compiler	Interpreter
Program can be run many times without the need to recompile	Source code can be run on any machine with the interpreter
Faster to execute	If a small error is found, no need to recompile the entire program
Executable code does not require the interpreter to run	
Compiled code cannot be easily read and copied by others	

Bytecode

- Most languages are not solely compiled or interpreted – they use a combination of both
- For example, Java is compiled into bytecode which is an intermediate step between source code and machine code
- The bytecode is interpreted by a bytecode interpreter, for example the Java virtual machine

Worksheet 4

- Do the questions in Task 1 on the worksheet

Stages of compilation

- A compiler goes through several stages to convert source code to object code
  - Lexical analysis
  - Symbol table
  - Syntax analysis
  - Semantic analysis
  - Code generation

Lexical analysis

- All unnecessary spaces and all comments are removed
- Keywords (e.g. print), constants and identifiers are replaced with tokens representing their function in the program

Lexical analysis

- For example look at the following code:

```
age = 17

print ( age )
```

- This might produce the following tokens:

```
<identifier> <operator> <number>
<keyword> <open_bracket> <identifier>
<close_bracket>
```

Symbol table

- The lexer will build up a symbol table for every keyword and identifier in the program
- The symbol table helps to keep track of the run-time memory address for each identifier

Item name	Kind of item	Type of item	Value	Pointer
age =			(memory address)	

Syntax analysis

- The stream of tokens from the lexing stage is split up into phrases
- Each phrase is parsed which means it is checked against the rules of the language
- If the phrase is not valid, an error will be recorded
- For example, this sequence of tokens may not be valid and this would be picked up by syntax analysis

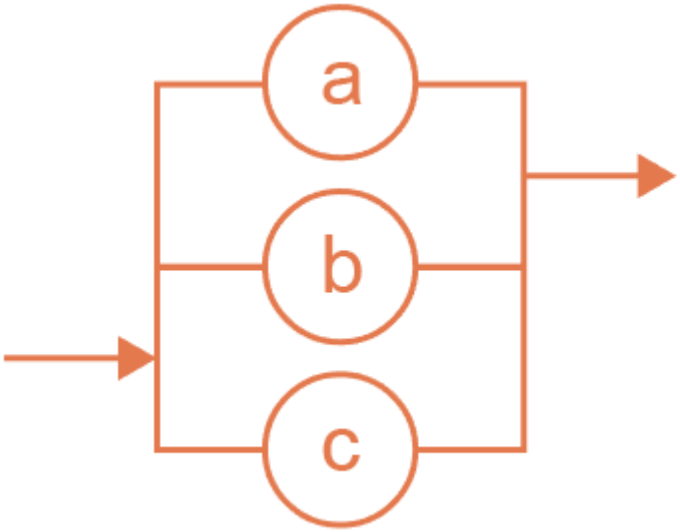
```
<number> <operator> <identifier>
```

(e.g. the source code might be 5 = a)

Syntax rules

- The rules of the language need to be defined
- Syntax rules can be drawn as a syntax diagram
  - Can you give an example of a word in this language?

Lette



Word

Worksheet 4

- Complete **Task 2** on **Worksheet 4**

Semantic analysis

- It is possible to create a sequence of tokens which is valid syntax but is not a valid program
- Semantic analysis checks for this kind of error
- For example this phrase may be valid syntax:

```
<if> <identifier> <operator>
<number>
```

(e.g. the source code might be: `if a > 5 )`

- …however if the identifier has not previously been declared then semantically it is not a valid program

Code generation

- Once the program has been checked, the compiler generates the machine code
- It may do this in several ‘passes’ over the code because code optimisation will also take place

Code optimisation

- Sometimes source code is written inefficiently
- Code optimisation aims to
  - Remove redundant instructions
  - Replace inefficient code with code that achieves the same result but in a more efficient way
- Can you think of any disadvantages that might result from code optimisation?

Worksheet 4

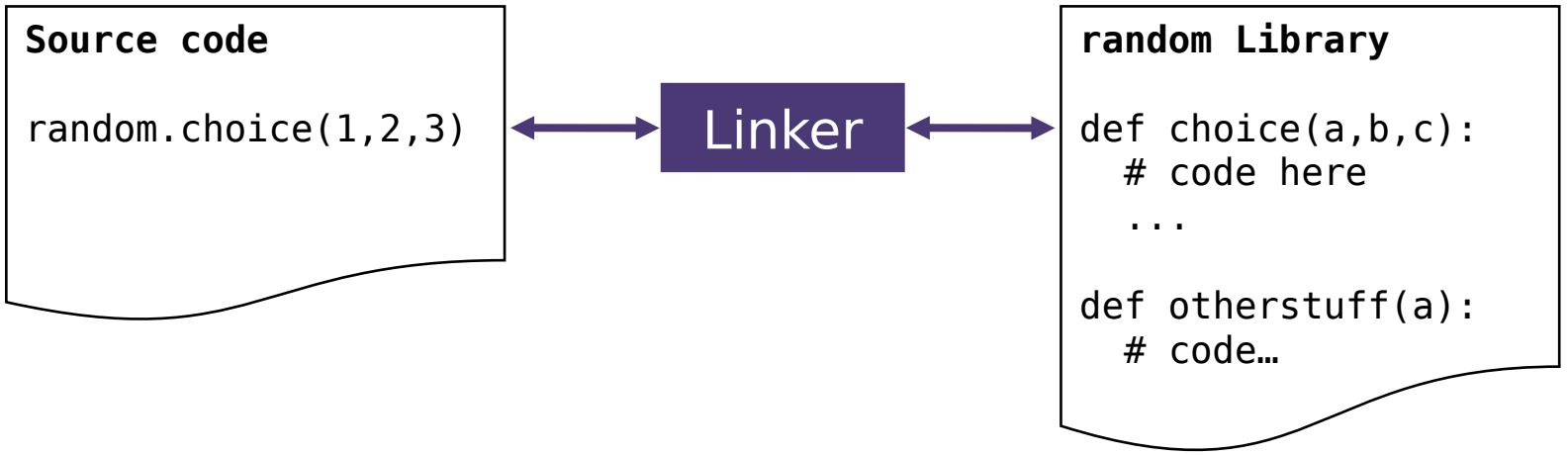
- Complete **Task 3** on **Worksheet 4**

Libraries

- Most languages have sets of pre-written (and pre-compiled) functions called libraries
- Examples could include functions for generating random numbers or for mathematical operations
  - Can you think of any other libraries you may have used?
- A programmer can also write their own libraries
- Library functions can be called within a program
  - What are the advantages of using libraries?

Linker

- The linker needs to put the appropriate memory addresses in place so that the program can call and return from a library function



Loader

- The job of the loader is to copy the program and any linked subroutines into main memory to run
- When the executable code was created it may assume the program will load in memory address 0
- However, memory addresses in the program will need to be relocated by the loader because some memory will already be in use

Plenary

- An assembler converts from low level code to machine code
- Compilers and interpreters convert from high level code to machine code
  - Compilers convert the whole program at once
  - Interpreters translate the program line by line
- Many languages convert source code to an intermediate stage called bytecode

Plenary

- When a program is compiled it is lexed, syntactically and semantically analysed, and optimised before executable code is generated
- Code from other files called libraries can be used
- Libraries are linked to the executable code by the linker
- The loader copies the executable code into main memory and adjusts memory addresses

Copyright

Licence agreement

PG ONLINE